Data Representation

COMP 1002/1402

Representing Data

- A computer's basic unit of information is:
  a *bit* (Binary digIT)
- An addressable memory cell is a *byte* (8 bits)
- Capable of storing one character

10101010

Counting Bytes

The normal meanings of kilo, mega & giga don't apply

1 kb	1 kilobyte	2^{10}=1024 bytes
1Mb	1 Megabyte	2^{20}=1024 kb
1 Gb	1 Gigabyte	2^{30}=1024 Mb

Computer Memory

RAM – Random Access Memory

Computer is an array of bytes each with a unique address

Address	Contents
A016	11010101
A017	00000000
A018	11001010
A019	00000001
A01A	10000001
A01B	10000010
A01C	10000011

Bits

All bits are numbered right to left:

For an 8 bit example:

10101010

76543210

Least signifcant bit (l.s.b.) is 0 & m.s.b. is 7

Information

All computer information is stored using bits

Characters	A a < , ! #
Integers	2, 3, 5, 7, 11, 13
Floating point nums	3.1415926535
Memory addresses	
Computer instructions	
Graphics information	Pictures, movies

Sequences of bits…

Everything is encoded into a sequence of bits

Q. How many sequences of n bits exist?

N	# of Sequences	Sequences
1	2	{0,1}
2	4	{00,01,10,11}
3	8	{000,001,010,011, 100,101,110,111}

Sequences of bits…

Mapping a bit sequence to an integer

N	# of Sequences	Range of integers
8	$\mathbf{2^{8}}$	256
16	$\mathbf{2^{16}}$	65,536
32	$\mathbf{2^{32}}$	4,294967296

Characters

ASCII – American Standard Code for Information Interchange

128 characters (7 bits required)

Contains:

- Control characters (non-printing)
- Printing characters (letters, digits, punctuation)

ASCII – Characters

Hex Equiv.	Binary	Character
00	00000000	NULL
07	00000111	Bell
09	00001001	Horizontal tab
0A	00001010	Line feed
0D	00001110	Carriage return
20	00100000	Space (blank)

ASCII – Characters

Hex Equiv.	Binary	Character
30	00110000	0
31	00110001	1
39	00111001	9
41	01000001	A
42	01000010	B
61	01100001	a
62	01100010	b

Alternative Representations

EBCDIC – Extended Binary Coded Decimal Interchange Code

Unicode – 16 bit Java code for many alphabets

Representations of Machine Instructions

- Every processor is different
- Families include: Intel, Motorola
- Instructions are stored in memory
- Machine code is a series of instructions

Motorola Example

Instruction	Operand	Address	Machine Code
LDA	#$30	3100	86
		3101	30
TFR	A, DPR	3102	1F
		3103	8B

How does the machine know?

Q. What is the difference between:

- A space (30 in hex)

And

- An Instruction (30 in hex)

Ans. Depends on how it is used.

Integer Representations

Remember machines are different!
Two possibilities:

- Unsigned Integers
  - only positive numbers
- Signed Integers
  - Negative numbers also allowed

Unsigned Integers

Representation of Unsigned integers in C:

Binary notation (different lengths possible)

8 bit (1 byte) **`unsigned char`**

Binary Rep.	Integer
00000000	0
00000111	7
10101010	170
11111111	255

Unsigned Integers

16 bit (2 bytes) **`unsigned int`**

Binary Rep.	Integer
0000000000000000	0
0000000000000111	7
0000000010101010	170
1111111111111111	65535

Signed Integers

- Several methods of representation
  - Signed Magnitude
  - Ones Complement
  - Twos Complement
  - Excess-M (Bias)

Signed Magnitude

- The sign is the leftmost bit
  - **0** is positive
  - **1** is negative

8 bits		16 bits	
00000111	***+7***	***0100000000000000***	***+16384***
10000111	***-7***	***1100000000000000***	***-16384***

Signed Magnitude

Issues:

- Simple
- Equal number of positive and negatives
- Two zero values (8 bit example)

  00000000 +0

  10000000 -0
- Two bit types = messy arithmetic

Signed Magnitude

Adding two numbers:

```
if (signs are same)
  add the two integers.
  result's sign is the sign of either.
else
  find the larger magnitude
  subtract the smaller from the larger mag.
  Result's sign is the sign of the larger mag.
end if
```

Complements

Eliminate the explicit sign!

Example is base 10 using 2 digits:

100 numbers possible (50 pos, 50 neg)

Define 00-49 as the positives

Complements

Rule for Addition:

- Add the numbers
- Any carry is discarded

e.g.:

```
  99    -01
  01    +01
 100     00
```

99	-1
98	-2
97	-3
96	-4

Complements

- Notation is consistent
- Get numbers without sign digit
- Used for negative numbers on a computer

Ones Complement

Complement with m.s.b. sign bit

- 0 is positive
- 1 is negative

Rule to change signs:

take the ones complement of the number

change 1’s to 0’s and vice versa

Ones Complement

8 bits		16 bits	
00000111	***+7***	***0100000000000000***	***+16384***
11111000	***-7***	***1011111111111111***	***-16384***

Ones Complement

Issues:

- Positives easy to interpret
- Negatives are difficult
  - Have to take complement to see magnitude

  11110000 = ? = -00001111 = -15
- Equal number of pos, neg
- Two zeroes (00000000 and 11111111)

Ones Complement

Rules for addition:

Add as usual for binary (include sign)

Add any carry to right side

Example:	+ 7	00000111
	- 12	11110011
	- 5	11111010

Ones Complement

Example:	+127	01111111
	- 63	11000000
		100111111
		1
	+64	01000000

Ones Complement

Overflow:

If the sum of two numbers is bigger or smaller than can be expressed

Example:	+127	01111111
	+ 1	00000001
	- 127	10000000

Ones Complement

Lastly Subtraction is simply addition with a sign change

127 – 5 = 127 + (-5)

Twos Complement

- Sign bit is leftmost bit
  - 0 is positive
  - 1 is negative
- Positives:
  - first bit is sign rest is binary

Rule to change sign:

Take ones complement and add one

(**never** subtract)

Twos Complement

8 bits		16 bits	
00000111	***+7***	***0100000000000000***	***+16384***
11111001	***-7***	***1010000000000000***	***-16384***

Two’s complement

Issues

- Positives are easy

- Negatives aren’t
  - Always take to twos complement
- Only one zero!!
- But what about largest negative

Two’s Complement

Rule for addition:

do normal binary and throw away carry!

Example:	+127	01111111
	- 10	11110110
	+117	101110101

Two's Complement

- What is the range of 4 bit 2's complement?

- Largest: 0111 = +7
- \- 7 = 1001
- -7 - 1 = -7 + (-1) = -8 or overflow

  -1 = 1111

Binary Coded Decimal

4 bits used to encode one decimal digit

$(4321)_{10}$ = 0100 0011 0010 0001

Promotes easy conversion to decimal

Easy to convert to ASCII equivalent

Binary Coded Decimal

Issues

- Wasted space
- 1 byte is two digits
- Computations are more complex

Excess-M (Bias)

Consider 8 bits

- Stores up to 256 items
- Half negative half positive (within one)

Use unsigned binary and pick middle (M)

10000000 (128) or 01111111 (127)

If M=127 then excess 127 = (value of int) - 127

Excess-M (Bias)

Stores exponent in Floating point rep.

11111111 in excess 127:	255 - 127 = 128
10000000	128 - 127 = 1
01111111	127 - 127 = 0
00000000	0 - 127 = -127

Excess-M (Bias)

Convert decimal to binary Excess-M:

Convert 19 to Excess-127

19+127 = 146 which is 10010010

Excess-M (Bias)

- Addition and other arithmetic is not easy

- Recognizing anything is difficult

- One zero

- Full range of numbers

Floating Point

Floating points represent fractional numbers

- Overall Sign
- Fractional Part / Mantissa
- Exponent
- Base

Floating Point

The type **`float`** in C:

Float   32 bits or 4 bytes

Sign	Exponent	Mantissa
<1 bit>	<-8 bits->	<- 23 bits ->

Sign is overall sign
Base is 2
Exponent is excess 127
Mantissa is 23 bits (about eight digits of accuracy)

Floating Point

<u>Example: </u> 01000000010000000000000000000000

sign bit =0   +
exponent = 10000000 = 1 in excess 127
mantissa 1 followed by zeros. We supply leading 1.

So number is $+(1.1)_2 * 2^1 = 1.5 * 2^1 = 3.0$

NOTE: Exception is 0.0 (all 0's)

Floating Point

What is the representation of -.875 * 2^{-5}?

Overall sign is "-" so bit =1

Mantissa = $(.111)_2$ =$(1.11)_2$ * 2^{-1}

Number is:

+$(1.11)_2$ * 2^{-1} * 2^{-5} = +$(1.11)_2$ * 2^{-6}

Exponent is -6 = -6 + 127 or 121 01111001

10111100111000000000000000000000

Floating Point

0.6 in floating point?

Subtract largest possible power of two:

= 0.5 + 0.1

= 0.5 + 0.0625 + 0.0375

= 0.5 + 0.0625 + 0.03125 + 0.00625

= 0.5 + 0.0625 + 0.03125 + 0.00390625 + 0.00234375 …

= 0.100110011 = 1.00110011*2^{-1}